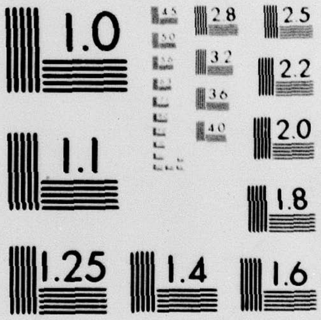
AD-A074 224   STANFORD UNIV CALIF STANFORD ELECTRONICS LABS   F/G 5/2
DESIGN AND VERIFICATION OF RELIABLE SOFTWARE.(U)
JUL 79 S S OWICKI   F49620-77-C-0045
UNCLASSIFIED   AFOSR-TR-79-0978   NL

| OF |

AD
A074224

END
DATE
FILMED
10-7
DDC


1.0
4.5
5.0
5.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

18 19 4

COMPUTER SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES
DEPARTMENT OF ELECTRICAL ENGINEERING

STANFORD UNIVERSITY

Stanford, California 94305

6

DESIGN AND VERIFICATION OF RELIABLE SOFTWARE

9

ANNUAL REPORT, no. 2 (Annual),

Covering the Period 1 Jan~~uary~~ — 31 Dec~~ember~~ 1978,

15

AFOSR Contract no. F49620-77-c-0045

SEL Project N-771

16 2304 17 A2

Prepared for the

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH

Bolling Air Force Base
Washington, D. C. 20332

Principal Investigator:

10

Prof. Susan S. Owicki

11 July 1979

12 11p.

Research Objectives

Program verification has frequently been suggested as a tool for improving software reliability. The purpose of this research is to develop verification techniques that can be applied to complex program systems involving concurrency, such as operating systems and network communications managers.

Two complementary approaches have been pursued. The first is the development of basic tools for describing and verifying concurrent programs; the aim is to find ways of reducing the complexity which arises from concurrent interactions between the components of the system. These tools are general purpose and should be applicable to a wide range of problems. The second approach is the investigation of specific applications by analyzing algorithms and programs from such domains as operating systems, networks, and distributed databases. These program examples provide test cases for evaluating the power of the basic tools and help to identify problem areas where further basic work is needed. In addition, analysis of these programs often makes it possible to recognize common patterns and formulate rules that simplify the design and verification of program components that fit these patterns.

Status of the Research Effort

1. Basic tools

Our purpose here is to devise general-purpose specification and proof methods for concurrent programs. Two types of correctness criteria can be distinguished: invariant properties, which should be true throughout system execution, and liveness constraints, which require that certain events must eventually occur. For example, an airline reservation system might preserve the invariant that no flight is overbooked and guarantee, as a liveness constraint, that each request for a reservation is eventually answered. Our techniques for dealing with the two kinds of problems are discussed below. In both cases, we are concerned with managing the complexity of the verification process, and so favor modular methods that allow factoring the verificaiton of a system into relatively independent treatment of each component.

First, let us consider the proof of invariant properties. Concurrent programs can be constructed using three kinds of modules: processes (the active components), monitors (which implement shared data objects and operations) and compound modules constructed from other modules. We have developed a specification format in which each module is described by assertions giving its initial state, its requirements from other modules, invariant relations between its local variables, and effects of its procedures (if it has any). Verification rules for proving that a module implementation meets its specifications take several forms. For process and monitor modules, verification requires direct analysis of the program code. Because of the specification style, this step is in general no more complex than for a non-concurrent program. For compound modules, verification depends only on the specification of the components, and not on their implementation. Since each module verification is performed independently, and the effect of interactions between components is limited to a small, well-defined interface, the complexity of the system proof does not grow unmanageably as the size of the system increases.

The basic rules for proving invariant properties are presented in [1]. In [2], the rules are amplified and illustrated by the design and verification of a simple system for routing mail in a ring network.

This method of specifying and verifying invariant properties is adequate for a wide range of concurrent programs. Our work on liveness properties is at an earlier stage, but the initial result are promising. Our approach is based on temporal logic, in which one can express assertions about future program states. This is accomplished by introducing two symbols: $\Diamond$(eventually) and $\Box$(henceforth). The liveness requirements of the airline reservations system described above can be expressed in temporal logic by:

$\Box$(request R received → $\Diamond$ request R answered).

Previously, the statement and verification of liveness properties has been done quite imformally. As a result, correctness proofs have been very susceptible to ambiguities in the problem statement and to errors of reasoning. The main contribution of temporal logic is that it provides a precise means of stating and proving liveness requirements.

The use of temporal logic in describing and verifying small programs is described in [6]. The methods presented there, however, are not suitable for large programs because they do not provide for modular decomposition. To facilitate modular proofs, we propose to extend the specifications of a module to include one or more conditional promises of the form

if P then $\Diamond$Q unless delayed by D,

where P, Q, and D are assertions about the program state.
As an example, consider a pipeline of processes communicating through buffers. Process P_i, which moves data from buffer B_{i-1} to buffer B_i, might have the following specifications:

1. $\Diamond B_{i-1}$ is not full unless delayed by B_i full
2. if x is in B_{i-1}, then $\Diamond f_i(x_i)$ is in B_i unless delayed by B_i full

Verifying that a module implemented by a process or monitor meets its liveness promises depends on axioms that characterize the liveness properties of programming language statements (the properties of synchronizing operations like semaphore wait and signal are especially important). For compound modules, the liveness of the whole module can be verified using just the specifications of its components. For example, in the pipeline program, the entire system of processes and buffers satisfies the unconditional promise.

$$\text{if } x \text{ in } B_0 \text{ then } \Diamond f_k\ (f_{k-1} \ldots (f_1\ (x) \ldots) \text{ is in } B_k$$

The proof of this unconditional system promise involves showing that P_i's promise to remove items from B_{i-1} cancels out P_{i-1}'s delay condition. Proof rules for verifying promises of the sort are currently being developed.

2. Applications

We have investigated the design and verification of concurrent systems in three applications areas: operating systems, network communications protocols, and distributed databases. The work in each area is described below.

In operating systems, we have considered the design of an operating system nucleus; most of this work is described in the first annual report. An important result was the identification of two patterns that account for most operating system modules: the transmitter, which produces a stream of output values from a stream of input values, and the resource allocator, which manages the sharing of some object between competing modules. Verification methods for these module types are described in [2] and [3].

Deadlock avoidance is a particular problem for resource allocators, and we have investigated it in some detail. A variety of deadlock-avoidance strategies have been proposed in the literature. They have in common the property that deadlock is prevented by refusing to allocate resource if doing so leads to an "unsafe" state: one in which further resource requests might result in deadlock. The strategies differ in the amount of computation they preform in evaluating a request and in the amount of information they require about future resource requests from the competing modules." In general, a strategy that performs more computation or uses more information about resource needs can recognize a larger number of requests as "safe", and so impose less delay on the competing modules. We have developed a model that allows a precise characterization and comparison of these differences; analysis of the model has suggested extensionsthat improve the performance of previous algorithms. This work is reported in [5].

The second application area we have studied is network communication protocols. Here most of the modules are instances of the same transmitter patter that occurs so often in operating systems. The principle difference is that communication protocols must function correctly even in the presence of processor failures and transmission line errors. We can incorporate these failure possibilities into the verification in a straightforward manner. For example, the invariant of a perfect communication link is that the output it has delivered is an initial segment of the input it has received. A communication link which may lose or re-order messages is characterized by an invariant that states that its output is a permutation of a subsequence of its input. The liveness promise of a failure-free link is that each input value will eventually be output, while the promise of a link which can lose a message a bounded number of times is that a message which appears a sufficient number of times in the input will eventually reach the output. We have found that our techniques for verifying invariants are satisfactory in this application. Work is in progress to find powerful means of dealing with liveness in order to cope with the liveness characteristics of unreliable modules.

The final application area we have considered is distributed databases. Here the issue of concern is maintaining the consitency of multiple copies of data while allowing access from several users to take place concurrently. A new consistency control algorithm has been developed [4]; it uses a distinguished "true copy" of each data item which is the locus of locking for that data item. The true copy may migrate throughout the system, and may be split into multiple "shared copies" that can be read but not modified. This allows concurrent operation, with a lower overhead of messages than many existing algorithms. Work is in progress to make the algorithms resilient in the face of failures. The consistency of the true-copy algorithm is easy to verify. It does not depend on the use of a particular policy for avoiding deadlocks or resolving competing resource demands, so it can serve as a basis for a variety of schemes whose consistency will then be easily verified.

The consistency criterion required of distributed databases is an example of an invariant property, which we are well equipped to verify. Proving liveness is more difficult, especially when reliable service in the presense of errors is required. It seems likely that the tools being developed for communication protocols will also be useful for distributed database systems.

Over all, our experience in examining applications problems has been fruitful. It confirms our belief that the basic tools for dealing with invariants are now satisfactory, although there is still work to be done in deriving techniques tailored to particular applications. In verifying liveness, we are at an earlier stage. The most pressing problem is to find methods of dealing with resilient systems that must provide service in spite of component failures.

Publications

[1]. Owicki, S. "Specifications and Proofs for Abstract Data Types in Concurrent Programs," in Bauer and Broy (ed.) Program Construction, Springer - Verlag, 1979, pp 174-197.

[2]. Owicki, S. "Specifications and Verification of a Network Mail System," in Bauer and Broy (ed.) Program Construction, Springer-Verlag, 1979, pp 198-234.

[3]. Owicki, S. "Verifying Parallel Programs with Resource Allocation," Proc. of the International Conference on Mathematical Studies of Information Processing, Springer-Verlag, to appear 1979.

[4]. Minoura, T. "A New Concurrency Control Algorithm for Distributed Database Systems," Proc. 4th Berkeley Conference on Distributed Data Management and Computer Networks, " August, 1979.

[5]. Minoura, T. "Deadlock Avoidance Revisited," submitted to Journal of the ACM.

[6]. Lamport, L. and Owicki, S. "Proving Properties of Concurrent Programs," in preparation, to be submitted to Computing Surveys.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 79-0978 — 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DESIGN AND VERIFICATION OF RELIABLE SOFTWARE	5. TYPE OF REPORT & PERIOD COVERED Interim
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Susan Owicki	8. CONTRACT OR GRANT NUMBER(s) F49620-77-C-0045
9. PERFORMING ORGANIZATION NAME AND ADDRESS Stanford University Computing Systems Laboratory Stanford, CA 94035	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Bolling AFB, Washington, DC 20332	12. REPORT DATE July 1979
	13. NUMBER OF PAGES 10
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

program verification, operating system design, parallel programming, concurrent programming, program proving, temporal logic, distributed databases, network protocols.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This research project is concerned with methods for developing provably correct system programs. Two complementary approaches have been pursued. The first is the development of basic tools for describing and verifying concurrent programs; the aim is to find ways of managing the complexity which arises from concurrent interactions between system components. Specification and proof methods for invariant properties of modular programs have been developed. They allow independent verification of each module, and provide a

20. Abstract continued.

sufficient basis for reasoning about a wide class of concurrent systems. Tools for stating and verifying other types of requirements, e.g. that a message is eventually delivered, are in an earlier stage. Techniques have been designed that use temporal logic, a logical system that allows statements about future events.

The second approach that has been taken here is the investigation of specific problems in three applications areas: operating systems, network communication protocols, and distributed databases. In addition to providing test cases for evaluating the basic tools, these applications are important in their own right. It has been possible to identify common patterns, and to build more sophisticated techniques for handling them from the basic tools discussed above. A major issue in these applications is dealing with unreliable components, and preliminary descriptions of methods for treating component failure are discussed.

Professional Personnel

Principle Investigator: Susan Owicki

Graduate Student Research Assistants:

Keith Marzullo	9/77 - present
Toshimi Minoura	9/77 - 1/78 and 10/78 - present
Alfred Spector	1/77 - 6/77

Interactions

A. Spoken Papers:

Owicki, S., "Verfiying Protocols as Parallel Programs" at Protocol Verification Workshop, March 20-21, 1979. Sponsored by DARPA, organized by Vint Cerf.

Owicki, S. Lecturer at International Summer School on Program Construction, Germany, July, 1978.

Owicki, S "Modeling Parallel Programs Using Temporal Logic" IFIP Working Group 2.2, Japan, August 1978.

B. Other interactions

Susan Owicki is co-principal investigator with John Hennessy on Joint Services Electronics Program contract DAAG29-79-C-0047 entitled "Realiability in Distributed Database Systems."

Susan Owicki was a member of the program committee for the International Symposuim on the Semantics of Concurrent Computation, Evian, France, July 1979.